man

$ man [1-9] *nomecomando*

man è il paginatore dei manuali del sistema

- Riceve in input il nome di un programma, di una utility o di una funzione
- Suddiviso in sezioni, ognuna delle quali contiene più di pagine differenti

Sezioni di man:

1. Programmi eseguibili e comandi della shell
2. Chiamate al sistema (funzioni fornite dal kernel)
3. Chiamate alle librerie (funzioni all'interno delle librerie di sistema)
4. File speciali (di solito trovabili in /dev)
5. Formati dei file e convenzioni (es. /etc/passwd)
6. Giochi
7. Pacchetti di macro e convenzioni (es. man(7), groff(7))
8. Comandi per l'amministrazione del sistema (solo per root)
9. Routine del kernel [Non standard]

Esempi

- `$ man man`
  il manuale del manuale!

- `$ man sleep`
  cerca in tutte le sezioni, restituisce la pagina della prima sezione trovata

- `$ man 1 sleep`
  cerca nella sezione 1, "Programmi eseguibili e comandi della shell"

- `$ man 3 sleep`
  cerca nella sezione 3, "Chiamate alle librerie (funzioni all'interno delle librerie di sistema)"

- `$ man 2 sleep`
  cerca nella sezione 2, nessun risultato trovato

- `$ man -a sleep`
  cerca in sequenza in tutte le sezioni, q per completare una sezione, enter per la successiva

- `$ man 2 chmod`
  cerca nella sezione 2, chiamate di sistema di basso livello fornite dal kernel

- `$ man 5 passwd`
  cerca nella sezione 5, formati dei file e convenzioni es. /etc/passwd

- `$ man 8 shutdown`
  cerca nella sezione 8, Routine del kernel [Non standard]

METACARATTERI DELLA SHELL

- #: considera ciò che segue come un commento

```
$ #prova di commento
```

- `var=val` assegna il valore val alla variabile var

```
$ var=5
```

- `$var` restituisce il valore della variabile di shell var

```
$ echo $var
```

- `${var}` come sopra, ma utile per migliorare la leggibilità

- `` `...` `` (il backquote si ottiene con alt+') esegue eventuali comandi contenuti tra gli apici inversi e li sostituisce con i relativi output

```
$ echo `date "+%Y-%m-%d"`        (Y=anno4cifre, m=[01-12], d=[01-31])
$ echo `date "+%D"`              (same as %m/%d/%y)
```

- `'...'` la stringa contenuta tra gli apici non viene interpretata dalla shell

```
echo 'date "+%D"'
```

- `"..."` solo alcuni metacaratteri vengono interpretati (in particolare: $, \ e gli apici inversi)

```
echo 'date $var'          (non viene interpretato)
echo "date $var"          (viene interpretato soltanto il $)
```

OPERATORI

`;` Eseguire più comandi in sequenza:

```
$ sleep 4 ; echo "Hello"
```

`&&` Eseguire i comandi solo se i precedenti hanno successo:

```
$ mkdir a && cd a
```

`||` Eseguire un comando solo se il precedente ha fallito:

```
$ (mv a.txt b.txt && echo 'fatto') || echo 'file inesistente'
```

REDIRECT

Di default i comandi Linux prendono l'input da tastiera (**standard input**) e mandano l'output ed eventuali messaggi di errore su video (**standard output**, **standard error**)

REDIRECT DI STANDARD OUTPUT

- `command > filename`
  memorizza l'output di command nel file filename

```
$ echo 'ciao' > a.txt
$ cat a.txt
```

- `command >> filename`
  aggiunge l'output di command in coda al file filename (append)

```
$ ls -l >> a.txt
$ cat a.txt
```

REDIRECT SELETTIVO DI STANDARD OUTPUT / STANDARD ERROR

Identificativo	Nome	Default
0	Standard input	Tastiera
1	Standard output	Terminale
2	Standard error	Terminale

```
#include <stdio.h>

int main() {
  printf("send to stdout\n");
  fprintf(stderr,"send to stderr\n");
  return 0;
}
```

```
$ gcc -o test test.c
$ ./test > out.txt 2> err.txt    (redirect di stdout e stderr su 2 file diversi)
$ cat out.txt
send to stdout
$ cat err.txt
send to stderr

$ ./test &> both.txt             (redirect di stdout e stderr su uno stesso file)
$ cat both.txt
send to stderr
send to stdout
```

REDIRECT DI STANDARD INPUT

- `command < filename`
  l'input di command è letto dal file filename

```
$ ls > a.txt
$ cat a.txt
$ wc –l < a.txt   (stampa il numero di newline in a.txt interpretandolo come stdin)

$ wc –l a.txt     (stampa il numero di newline in a.txt interpretandolo come file,
                   quindi aggiunge anche il nome del file dopo il risultato [provare ad
                   esempio wc -l a.txt b.txt per vedere più output seguiti dal filename])
```

PIPE

Operatore | connette l'output di un programma all'input di un altro

Il flusso di esecuzione è da sinistra verso destra

```
$ ls –l | less
```

equivale a:

```
$ ls –l > tmp
$ less < tmp
```

Esempi

```
$ ls > elenco.txt          (file contenente i nomi dei file nella dir corrente)
$ ls >> elenco.txt         (duplico i nomi appendendo in coda)
$ sort elenco.txt          (ordine alfabetico)
$ sort elenco.txt | uniq   (oridno e rimuovo le ripetizioni)


$ cat elenco.txt | head –2                (mostro le prime due righe)
$ cat elenco.txt | head –2 | tail –1      (ultima riga delle prime due)


$ >> f1                    (creo 1 file)
$ cp f1 f2                 (lo duplico)
$ tar cf A.tar f1 f2       (creo un archivio, non compresso)
$ tar tvf A.tar            (verifico il contenuto)
$ gzip A.tar               (comprimo)
```

oppure con un unico comando, reindirizzando i flussi SENZA creare file intermedi:

```
$ >> f1 && cp f1 f2 && tar cvf – f1 f2 | gzip > A.tar.gz
```